

otelcol receiver 自作RTA

id:arthur-1 株式会社はてな

2024-05-09 Pepabo Tech Conference #22 春のSREまつり LT



Arthurと申します

株式会社はてな
アプリケーションエンジニア

ポケカの練習仲間が欲しい

X: @Arthur1__



Mackerel作ってます



監視を育てる、
「Mackerel」

クラウド時代に最適な監視モデルを
使いやすいUIで提供し、
システムの運用・監視に
チームで取り組む文化を作る
「クラウド運用の道標」となる
SaaS型サーバー監視サービス。

無料トライアルをはじめる

mackerel-bizde...
Dashboards
Overview
Blog
sample
参考URL
遠次ミーティング
Hosts
Services
Blog
small-go-webapp
Monitors
Channels
Alerts
導入マニュアル
新規ホストの登録

Blog
30min 1h 6h 1d 1W 1M 1Y
0.0787
0.0728
0.417
0.427
0.244
web
loadavg5
worker

OTel対応もやってます



時間がないので早速



レギュレーション

- OpenTelemetry CollectorのReceiver (Scraper) を作る
 - Scraper: 一定間隔でデータを取りに行く
- **sample.ultimate_answer**という名前で、常に42という値を持つ**メトリック**を作る
 - configで渡したsample.answererを属性として持つ
- 任意のexporterでメトリックが確認できたらタイマーストップ
- LTの時間だと少々無理があるのでつくりおきOK
 - RTAというより3分クッキング



チャート

- metadata.ymlを記述
- mdatagenでコード生成
- configを実装
- scraperを実装
- factoryを実装
- debug exporterと組み合わせてbuildして動作確認



以降の資料は投影しません
Xの **#pepabotech** で
投稿したものを参照ください



ディレクトリの準備

Componentごと階層を切る習慣がある
sampleという名前の自作Receiverを作るなら
receiver/samplereceiver という感じ

```
$ mkdir receiver/samplereceiver
```

```
$ go mod init samplereceiver
```



mdatagenのインストール

mdatagen: メタデータを記述したYAMLからCollectorに関するコード生成をしてくれるCLIツール

v0.100.0現在、通常のgo install・go runが失敗する ([#9281](#))
ため、ソースコードを手元に用意してinstallしよう

```
$ git clone git@github.com:open-telemetry/opentelemetry-collector.git
```

```
$ cd opentelemetry-collector/cmd/mdatagen
```

```
$ go install .
```

```
$ mdatagen -h
```



metadata.yaml

```
type: sample

status:
  class: receiver
  stability:
    development: [metrics]
  codeowners:
    active: [Arthur1]

attributes:
  sample.answerer.name:
    type: string
    description: Answerer name
```

```
metrics:
  sample.ultimate_answer:
    enabled: true
    description: Ultimate Answer
    unit: 1
    gauge:
      value_type: double
    attributes: [sample.answerer.name]
```

レギュレーションに書かれていた
内容を大体ここで定義している



metadata.yamlのスキーマ

(少なくとも私の環境では) GCP Blueprint Metadataファイルだと誤検知されて、エラーが出まくって困る

GCP Blueprint Metadata - Blueprint Solutions for Google Cloud (gcp-blueprint-metadata.json)

1 # Sample metadata file with all available configurations for a rec



2
3 Property sem_conv_version is not allowed. yaml-schema: GCP
4 Blueprint Metadata

5 View Problem (⇧F8) No quick fixes available

6 sem_conv_version: 1.9.0

7

sample



metadata.yamlのスキーマ

otelcolのリポジトリでオレオレフォーマットのスキーマは公開されているが、これではエディタの支援を受けられなくて困る

そこで、自家製JSON Schemaをつくりおきしておきました

<https://github.com/Arthur1/otelcol-metadata-schema>

```
# yaml-language-server: $schema=https://raw.githubusercontent.com/Arthur1/otelcol-metadata-schema
```

```
type: sample  
scope_name: go.opentelemetry.io
```

OTel Semantic Conventions version that will be associated with the scraped metrics. This attribute should be set for metrics compliant with OTel Semantic Conventions.

Source: [metadata.schema.json](#)

```
sem_conv_version: 1.9.0
```



コード生成

generate.goを作成してコマンドを記述

go generate ./... でコード生成

直接mdatagenを叩くと生成コードのpackage名が定まらないから
エラーになる

```
package samplerreceiver
```

```
//go:generate mdatagen metadata.yaml
```



生成結果

テストコード内でNewFactoryがないよと怒られる
→ これを自分で実装しないといけない

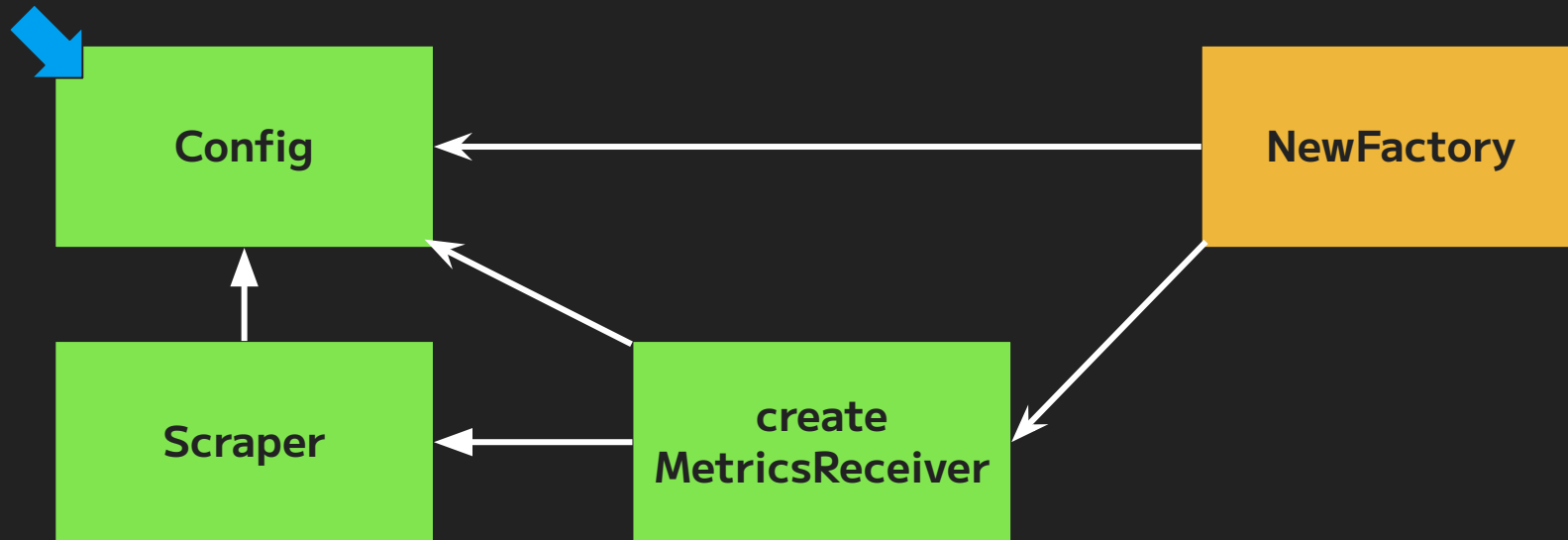
```
13     "go.opentelemetry.io/collector/consumer/consumert
14     "go.opentelemetry.io/collector/receiver"
15     "go.opentelemetry.io/collector/receiver/receivert
16 )
17
18 func TestCo
19     require.Equal(t, "sample", NewFactory().Type().St
20 }
21
```

run test | debug undefined: NewFactory compiler(UndeclaredName)
View Problem (⌘F8) No quick fixes available



NewFactoryには何が必要か

こういう依存関係なので、まずConfigから作ろう



Configの実装

要はotelcolのconfigファイルに何を書かせてどう読むかという定義

structを定義して、Default Configを生成する関数を作る

```
type config struct {
    scraperhelper.ControllerConfig `mapstructure:",squash"`
    metadata.MetricsBuilderConfig `mapstructure:",squash"`
    AnswererName                   string `mapstructure:"answerer_name"`
}
```

```
func defaultConfig() component.Config {
    cc := scraperhelper.NewDefaultControllerConfig()
    mbc := metadata.DefaultMetricsBuilderConfig()
    return &config{
        ControllerConfig:    cc,
        MetricsBuilderConfig: mbc,
        AnswererName:        "",
    }
}
```



Scraperの実装

```
type sampleScraper struct {  
    cfg *config  
    mb  *metadata.MetricsBuilder  
}
```

metadata.yamlから生成したコードに
特定のメトリックを作る関数が含まれている

```
func newScraper(cfg *config, settings receiver.CreateSettings) *sampleScraper {  
    return &sampleScraper{  
        cfg: cfg,  
        mb:  metadata.NewMetricsBuilder(cfg.MetricsBuilderConfig, settings),  
    }  
}
```

```
func (s *sampleScraper) scrape(ctx context.Context) (pmetric.Metrics, error) {  
    ts := pcommon.NewTimestampFromTime(time.Now())  
    s.mb.RecordSampleUltimateAnswerDataPoint(ts, float64(42), s.cfg.AnswererName)  
    return s.mb.Emit(), nil  
}
```



createMetricsReceiverの実装

receiver.CreateMetricsFuncという型を満たす関数を作る

先ほど作ったScraperをwrapするイメージ

```
func createMetricsReceiver(  
    _ context.Context, settings receiver.CreateSettings,  
    cfg component.Config, consumer consumer.Metrics,  
) (receiver.Metrics, error) {  
    c, ok := cfg.(*config)  
    if !ok {  
        return nil, fmt.Errorf("error")  
    }  
    s := newScraper(c, settings)  
    scraper, err := scraperhelper.NewScraper(metadata.Type.String(), s.scrape)  
    if err != nil {  
        return nil, err  
    }  
    return scraperhelper.NewScraperControllerReceiver(  
        &c.ControllerConfig, settings, consumer, scraperhelper.AddScraper(scraper),  
    )  
}
```



NewFactoryの実装

ようやく当初の目的のNewFactoryにたどり着いた

```
func NewFactory() receiver.Factory {  
    return receiver.NewFactory(  
        metadata.Type,  
        defaultConfig,  
        receiver.WithMetrics(  
            createMetricsReceiver,  
            metadata.MetricsStability,  
        ),  
    )  
}
```

ここまでできたら、生成されたテストコードを動かして
通ることを確認しよう 



ocbのインストール

Receiverの実装はこれにて完結

次はOTel Collectorに作ったReceiverを混ぜてビルドしていく

ocb (builder): OTel CollectorをビルドするためのCLIツール

YAMLを書いてコード&バイナリ生成

```
$ go install go.opentelemetry.io/collector/cmd/builder@latest
```

```
$ builder version # コマンドの名前がデカすぎて固定資産税かかりそうだな！
```

```
ocb version v0.100.0
```



ocbのYAMLを書く

receiverに先ほど作ったpackageを、exporterにdebugexporterを指定
ローカルのコードを参照するのでreplaces (go.modのreplaceディレク
ティブに相当) を忘れずに

```
dist:
  module: my-opentelemetry-collector
  output_path: ./
  version: 0.0.0
receivers:
  - gomod: samplereceiver v0.0.0
exporters:
  - gomod: go.opentelemetry.io/collector/exporter/debugexporter v0.100.0
replaces:
  - samplereceiver => ./receiver/samplereceiver
```



ビルド

go.modやmain.goなどがコード生成され、
最終的にotelcol-customというバイナリができる

```
$ # Receiverのコーディングはもう終わってるのでプロジェクトのルートに戻る
$ test -d receiver/samplereceiver
$ builder --config ./builder-config.yaml
2024-05-07T09:56:29.124+0900 INFO internal/command.go:125
OpenTelemetry Collector Builder {"version": "", "date": "unknown"}
...
$ test -f ./otelcol-custom
```



実行

ここまで来たらもう少し
(残りは利用者目線)

otelcolのconfigを書いて
実行しよう

```
receivers:
  sample:
    answerer_name: arthur-1
    collection_interval: 30s
exporters:
  debug:
    verbosity: detailed
service:
  pipelines:
    metrics:
      receivers: [sample]
      exporters: [debug]
```

```
$ ./otelcol-custom --config otelcol-config.yaml
```



実行結果

debugexporterは、メトリックの中身をログに出力してくれる

```
~/Repositories/private/suburi >>> ./otelcol-custom --config ./otelcol-config.yaml
2024-05-07T10:18:06.092+0900 info service@v0.100.0/service.go:102 Setting up own telemetry...
2024-05-07T10:18:06.094+0900 info service@v0.100.0/telemetry.go:103 Serving metrics {"address": ":8888", "level": "Normal"}
2024-05-07T10:18:06.095+0900 info exporter@v0.100.0/exporter.go:275 Development component. May change in the future. {"kind": "exporter", "data_type": "metrics", "name": "debug"}
2024-05-07T10:18:06.098+0900 info receiver@v0.100.0/receiver.go:310 Development component. May change in the future. {"kind": "receiver", "name": "sample", "data_type": "metrics"}
2024-05-07T10:18:06.099+0900 info service@v0.100.0/service.go:169 Starting otelcol-custom... {"Version": "0.0.0", "NumCPU": 10}
2024-05-07T10:18:06.099+0900 info extensions/extensions.go:34 Starting extensions...
2024-05-07T10:18:06.099+0900 info service@v0.100.0/service.go:195 Everything is ready. Begin running and processing data.
2024-05-07T10:18:06.099+0900 warn localhostgate/featuregate.go:63 The default endpoints for all servers in components will change to use localhost instead of 0.0.0.0 in a future version. Use the feature gate to preview the new default. {"feature gate ID": "component.UseLocalHostAsDefaultHost"}
2024-05-07T10:18:07.102+0900 info MetricsExporter {"kind": "exporter", "data_type": "metrics", "name": "debug", "resource metrics": 1, "metrics": 1, "data points": 1}
2024-05-07T10:18:07.102+0900 info ResourceMetrics #0
Resource SchemaURL:
ScopeMetrics #0
ScopeMetrics SchemaURL:
InstrumentationScope samplerreceiver 0.0.0
Metric #0
Descriptor:
  -> Name: sample.ultimate_answer
  -> Description: Ultimate Answer
  -> Unit: 1
  -> DataType: Gauge
NumberDataPoints #0
Data point attributes:
  -> sample.answerer.name: Str(arthur-1)
StartTimestamp: 2024-05-07 01:18:06.098511 +0000 UTC
Timestamp: 2024-05-07 01:18:07.102176 +0000 UTC
Value: 42.000000
{"kind": "exporter", "data_type": "metrics", "name": "debug"}
```



ここでタイマーストップ



完走した感想を話す



これまで作った自作receiver

<https://github.com/Arthur1/opentelemetry-collector-arthur1> リポジトリで公開中

- runnreceiver
 - runnで実行したシナリオテストの結果をメトリックとして送る
- cloudflaremetricsreceiver
 - Cloudflareの分析メトリック（ゾーンへのHTTPリクエスト数とか）を収集する



THE END

最後までご清聴いただき
ありがとうございました

